Chapter 9

Simulation of constrained dynamical systems

9.1 Introduction

SIMULATION offers a way to gain insight in the behaviour of a dynamical system also in the case where an analytical solution is not available. Digital simulation of a constrained continuous-time mechanical system is known to be liable to numerical instabilities. Our investigations showed that the cause of these instabilities is closely related to representation issues. For this reason we enclose the results of our research in this thesis. Most of the results presented in this chapter have appeared in 'A.A. ten Dam, Stable numerical integration of dynamical systems subject to equality state-space constraints, Journal of Engineering Mathematics, Vol. 26, pages 315-337, 1992'. The numerical method derived here has been applied successfully in for instance [33, section 3] and [53, 148] to simulate flexible robotic manipulators.

In this chapter we will discuss one particular issue in numerical simulation of continuous-time dynamical systems: obtaining a discretized model for a class of constrained dynamical systems. For unconstrained dynamical systems there are many discretization methods available. Discretization formulas for linear systems can be found in [77]. For nonlinear systems there are many numerical methods available to solve Ordinary Differential Equations (ODEs) [2, 134]. This is not the case for Differential Algebraic Equations (DAEs): establishing solvers for DAEs is still a very active research area. Overviews of the theory can be found in [21, 68]. In this chapter we will exploit the structure of a mathematical model of a constrained mechanical system to derive a numerical method that can be used for stable simulation of this class of systems. The numerical method aims at supporting real-time simulation [42]. Since real-time simulation often involves trade-off between accuracy and computation time, we will introduce a number of parameters that are available to the user of a simulation program to costumize simulation to its specific needs.

The remainder of this chapter is as follows. We will first discuss bilaterally constrained mechanical systems. In section 9.2 we will review briefly some of the numerical methods available in the literature. An example is discussed to show the existence of the instability phenomenon. In section 9.3 the instability phenomenon is analysed. We will derive a technique which solves the instability problem for a large class of systems. In section 9.4 we apply the results to the class of bilaterally constrained mechanical systems. Simulation results are presented. In section 9.5 we will briefly discuss simulation of unilaterally constrained mechanical systems. Conclusions can be found in section 9.6.

9.2 Conventional solution methods

In the present literature on numerical integration DAEs are often characterized by their index [21, 27, 68]. Roughly speaking, the index equals the number of times the constraints must be differentiated to arrive at a set of ordinary differential equations. The index can be viewed upon as a measure of how far a DAE is from being an ODE. A popular DAE-solver is DASSL. Designed by Petzold in the early eighties, DASSL is capable of solving DAEs which have a low index [118]. Constrained mechanical systems often have an index equal to three [21]. It is well known that index three systems can not be solved directly by standard ODE solvers [52, 63]. For constrained mechanical systems, stable numerical algorithms are few, and are usually available as research code only. Recently, an extension of DASSL has been used successfully in [60] to simulate several discontinuous phenomena that affect robot motion (see section 9.5). In the remainder of this section we will show that an instability phenomenon is present when simulating constrained mechanical systems with standard ODE-solvers.

Recall from proposition 8.4.1 (*ii*) the equations for a bilaterally constrained mechanical system

$$\begin{aligned} M(y)\frac{d^2y}{dt^2} + N(y, \frac{dy}{dt}) &= v + G^T(y)\lambda, \\ \phi(y) &= 0, \end{aligned} \tag{9.1}$$

with $\phi : \mathbb{R}^d \to \mathbb{R}$, as a model for a bilaterally constrained mechanical system. Here $G(y) := \frac{\partial \phi}{\partial y}(y)$ is assumed to have full row-rank. This implies that system (9.1) has relative degree $r_0 = 2$. Differentiating the constraint equation gives

$$G(y)\frac{dy}{dt} = 0. \tag{9.2}$$

This equation is referred to as a hidden constraint for a constrained mechanical system. Let $\mathcal{M}_0$ be a neighbourhood of $(x_{1,0}, x_{2,0})$ where $\phi(x_{1,0}) = 0$ and $\frac{\partial \phi}{\partial x_1}(x_{1,0})x_{2,0} = 0$. From

(7.15) we obtain that the largest controlled invariant manifold $\mathcal{N}^*$ is given by:

$$\mathcal{N}^* = \{(x_1, x_2) \in \mathcal{M}_0 \,|\, \phi(x_1) = 0, G(x_1)x_2 = 0\}. \tag{9.3}$$

The equations above provide the basic ingredients to find a numerical solution of a constrained mechanical system. The conventional way to do this is by first solving the system of equations in (9.1) for λ. This leads to the formulation in proposition 8.4.1 (*iii*). This model contains an explicit expression of the Lagrange multiplier. Secondly, this model is discretized. Thirdly, the resulting set of equations is used to progress the solution in time with the aid of a numerical algorithm. This sequence of steps exhibits stability problems. This can be shown by performing simulations where the discrete equations are obtained in the conventional manner. We will use a simple example from [38].

Example 9.2.1 *A ball constrained by a circular basin.* Consider the ball of example 1.2.3. The initial conditions of the ball are chosen such that the constraints are not satisfied, but the initial position is close to $(y_1(0), y_2(0)) = (1, 0)$. (Details with respect to this simulation is given in section 9.4.) Figure 9.1 depicts the result of a simulation. The motion of the

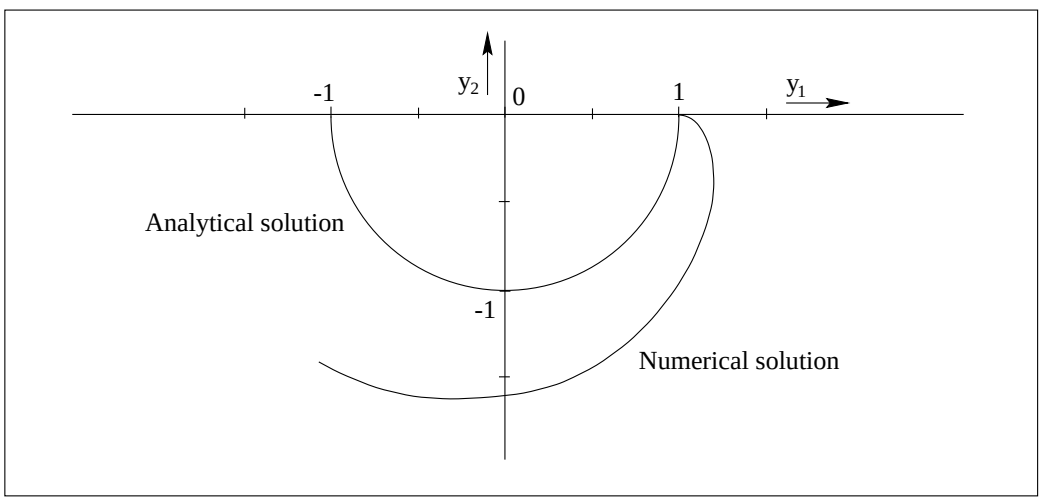

Figure 9.1: Erroneous solution of a bilaterally constrained dynamical system.

ball should be on the part of the unit circle that is shown in the figure. It can be seen that the simulated behaviour does not satisfy the position constraint. Moreover, the position constraint violation increases with time. ◁

During digital simulation there will be other sources of error apart from errors in the initial conditions. We mention discretization errors, errors due to finite-word-length, and integration errors. The collection of these errors will be referred to as *numerical errors* in the remainder. The presence of these numerical errors leads to violation of the constraints and eventually leads to a drift-off from the constraint manifold. As a consequence one obtains physically meaningless solutions. (The initial conditions in example 9.2.1 have

been chosen to visualize this drift-off, or instability phenomenon.) Usually, for consistent initial conditions, it will take a longer period of time for the drift-off to become noticeable. And if the drift-off remains small then the corresponding approximate solution may well be acceptable. But generally speaking, a growing drift-off can not be accepted. Another striking example of drift-off can be found in [52].

Example 9.2.1 illustrates that numerical simulation of the conventional formulation of a constrained mechanical system exhibits severe stability problems already for simple systems. Simulation of multibody systems is an activity with a long history. As a result several computational procedures have been proposed to overcome the stability problems. These include techniques where a distinction is made between dependent and independent variables (a solution is sought through singular-value-decomposition), equilibrium correction strategies [8], penalty formulations [10, 115], coordinate partitioning methods [141], predictor/corrector algorithms [128], a differential algebraic approach [64], and projection methods [52].

In engineering practice, the constraint stabilization technique presented by Baumgarte [8] is often applied because it is conceptually simple and it is easy to implement. Differentiating the position constraint in equation (9.1) twice gives

$$\frac{d^2\phi(y)}{dt^2} = 0. \tag{9.4}$$

It is well known that the numerical solution of (9.4) can be unstable, i.e. it can lead to values of $\phi(y)$ and $\frac{d\phi(y)}{dt}$ that are far from the desired value zero. The modified acceleration equation:

$$\frac{d^2\phi(y)}{dt^2} + 2\alpha\frac{d\phi(y)}{dt} + \beta^2\phi(y) = 0, \tag{9.5}$$

is (asymptotically) stable for $\alpha > 0$ [149]. The additional terms in (9.5) can be seen to act as a proportional/derivative control with gains equal to 2α and β^2. Baumgarte also presented the proportional/integral counterpart, for the asymptotic stabilization of holonomic constraints [9]. One problem can readily be seen from the formulation of the stabilization technique: how to choose the coefficients α and β? Since the stabilization term can be interpreted as a proportional/derivative control law, it is noted that the use of the stabilization term shifts the poles of the system and alters the dynamic behaviour of the system. The choice of α and β is merely a matter of how fast we want to damp out the constraint violations. Large values of α and β lead to high-gain feedback laws. Note that the choice $\alpha = \beta$ yields a critically damped system. It is this choice that is commonly used when Baumgarte's method is applied. In [30] the gains in (9.5) are related to the step size that is applied in the numerical algorithm. There it is remarked that their choice of gains tends to damp out constraint violations faster than any other choice, but accumulation of (integration) errors can not be prevented. Furthermore, decreasing the step size results in larger gains. As a result the damping terms dominate the numerical solution process of (9.5):

they make the system become numerically stiff. A further analysis of Baumgarte's method can be found in [5]. In spite of these drawbacks, the constraint stabilization technique of Baumgarte is often applied since it avoids iterative solution of algebraic constraints. This is in contrast to for instance a predictor/corrector algorithm and some of the other methods. For instance the projection method proposed in [52] uses a combination of the numerical solvers known as Backward Difference Formulas (BDFs) and a Gauss-Newton projection method. These algorithms require iteration processes to obtain values within a certain predefined error level: a number of corrector steps must be applied. Iteration processes are known to increase computation time considerably.

Simulation of complex systems that involve human interaction is directed towards real-time simulation. In general this puts very stringent requirements on hardware, tool software and model software [42]. In some cases it may require years of close cooperation by experts from different fields of expertise to arrive at a validated real-time model, see for instance [148]. It is clear that from a computation-time perspective, numerical methods that do not use iteration processes have an advantage over numerical methods that do use these processes, not withstanding the success of the latter methods. As remarked before, the selection of a method also involves a trade-off between accuracy and computation time. We will propose a method that can be used in real-time simulation studies of complex systems. We restrict ourselves to numerical integration by well-known explicit ODE-solvers. It can be seen, however, that the basic idea can be extended easily to for instance a BDF method.

9.3 Analysis of the instability phenomenon

Since all continuous-time formulations in proposition 8.4.1 are equivalent, any one of them can be chosen to find the time-evolution of the DAE. In the formulation given in proposition 8.4.1 (*iii*) the constraint equations are not explicitly present. Indeed, it is this formulation that is often used to obtain a discrete-time set of equations. Unfortunately, this is not a good basis in our search for a numerical solution, as will be demonstrated next. The analysis will be done in the general setting of section 7.6, for the case $r_0 = 1$. The following assumption is made.

Assumption 9.3.1 *Simulation assumption I.* $\exists \delta > 0$ such that $|h(x)| \leq \delta \Rightarrow L_g h(x)$ has full row-rank.

Assumption 9.3.1 (*i*) is an extension of assumption 7.2.2 (*i*) when $r_0 = 1$. We will also assume throughout this chapter that the time-step is chosen such that the unconstrained system can be simulated in a stable manner. This means that the time-step depends on the numerical method under consideration, but we will, to shorten notation, denote the time-step simply by Δt.

Apply the Forward-Euler (FE) integration method to the formulation in proposition 7.6.1 (*iii*). There holds $C(x) = L_g h(x) = G(x)g(x)$. To shorten notation let the subscript n denote the value of a time-sequence at time $t_n := n\Delta t$, $n \in \mathbb{Z}_+$. It follows that for the numerical approximations one has

$$\begin{aligned} x_{n+1} &= x_n + \Delta t f(x_n) + \Delta t(g(x_n)v_n + g(x_n)Z(x_n)C^T(x_n)\lambda_n), \\ \lambda_n &= -(C(x_n)Z(x_n)C^T(x_n))^{-1}(L_f h(x_n) + C(x_n)v_n). \end{aligned} \tag{9.6}$$

Let ϵ_n denote the error made in the calculation of λ_n at time t_n. Then the position constraint-violation at time t_{n+1} can be calculated as follows. With neglect of the higher-order terms we obtain from (9.6)

$$\begin{aligned} h(x_{n+1}) &\overset{FE}{=} h(x_n + \Delta t(f(x_n) + g(x_n)v_n + g(x_n)Z(x_n)C^T(x_n)\lambda_n)) \\ &\overset{Taylor}{=} h(x_n) + G(x_n)\Delta t(f(x_n) + g(x_n)v_n + g(x_n)Z(x_n)C^T(x_n)\lambda_n) \\ &\overset{(9.6)}{=} h(x_n) + \Delta t G(x_n)(f(x_n) + g(x_n)v_n) \\ &\quad -\Delta t G(x_n)g(x_n)Z(x_n)C^T(x_n) \cdot \\ &\quad ((C(x_n)Z(x_n)C^T(x_n))^{-1}(G(x_n)f(x_n) + G(x_n)g(x)v_n) - \epsilon_n) \\ &= h(x_n) + \Delta t C(x_n)Z(x_n)C^T(x_n)\epsilon_n. \end{aligned} \tag{9.7}$$

We have obtained a recursive expression that can be rewritten as:

$$h(x_{n+1}) = h(x_0) + \Delta t \sum_{i=0}^{n} C(x_i)Z(x_i)C^T(x_i)\epsilon_i. \tag{9.8}$$

From the latter equation it follows that once an error is made in the calculation of the Lagrange multiplier the solution is not on the constraint manifold. Now consider the special case where $\epsilon_n = \epsilon, \forall n \in \mathbb{N}$, with ϵ constant, and that C and Z are constant matrices. This gives

$$h(x_{n+1}) = h(x_0) + t_n\, CZC^T\epsilon. \tag{9.9}$$

Note that it makes no sense to let $\Delta t \to 0$. And if $t_n \to \infty$, for instance because we are interested in an equilibrium solution, one even has that $h(x_{n+1}) \to \infty$! The analysis also shows the importance of correct initial conditions. But even if the initial conditions are consistent with the equality constraints, error amplification is inevitable due to the presence of numerical errors. Each error source will contribute to the drift-off. Our analysis reveals that once an error is made in the solution for the generalized Lagrange multiplier error amplification can not be prevented if the formulation in proposition 7.6.1 (*iii*) is used.

We will now show that if we start with the formulation in proposition 7.6.1 (*ii*) a numerical method can be obtained that has the property that it is robust with respect to errors in the initial conditions, and stable with respect to errors made during numerical integration.

Opposed to the sequence of steps that is applied in the previous section, we propose a different sequence. This sequence of steps can be described as *discretize first - substitute next - combine later*. This approach to simulation has been applied to restricted ODEs in [38] and to boundary value problems of Partial Differential Equations (PDEs) in [138]. But the original idea can already be found in [74] where it is applied to index one systems with linear, stationary constraints in combination with the Forward-Euler integration method.

First the equations in proposition 7.6.1 (*ii*) are discretized using the Forward-Euler method. This gives

$$x_{n+1} = x_n + \Delta t(f(x_n) + g(x_n)v_n + g(x_n)Z(x_n)C^T(x_n)\lambda_n^d), \tag{9.10}$$

$$h(x_{n+1}) = 0. \tag{9.11}$$

Note that we now treat the constraint in an implicit manner. The notation λ_n^d is used to distinguish the *discrete Lagrange multiplier* in (9.10) from the one in (9.6). The idea now is to obtain a discrete formula for λ_n^d directly from (9.10) and (9.11), rather than using the continuous-time expression in proposition 7.6.1 (*iii*). This implies that the expression for the discrete generalized Lagrange multiplier depends on the integration method used to find the time-evolution of a DAE. However, we will show that one single expression for the discrete Lagrange multiplier can be used in combination with a number of different integration methods. It will be useful to derive an expression for λ_n^d for the case where matrices C, G and Z are constant.

Lemma 9.3.2 Let the assumptions made in proposition 7.6.1 hold for constant matrices C, G and Z. Then with the Forward-Euler integration method the discrete generalized Lagrange multiplier λ_n^d is given by

$$\lambda_n^d = -(CZC^T)^{-1}\left(G(f(x_n) + g(x_n)v_n) + \frac{h(x_n)}{\Delta t}\right).$$

If we compare the expression for λ_n in (9.6) with the expression for λ_n^d in lemma 9.3.2 we have ◁

$$CZC^T\lambda_n^d = CZC^T\lambda_n - \frac{h(x_n)}{\Delta t}. \tag{9.12}$$

Due to numerical errors, $h(x_n) \neq 0$. As a result we have that the equivalent continuous-time formulations of proposition 7.6.1 do not give equivalent discrete-time formulations.

By use of the appropriate continuous-time formulation, i.e. the formulation in which the constraint is still present, stable numerical integration can be attained. We already showed that with the conventional Lagrange multiplier λ_n error accumulation can not be avoided. Performing the same analysis that lead to (9.9) but now with λ_n^d instead of λ_n gives (for

constant C, G, Z and ϵ)

$$h(x_{n+1}) = \Delta t\, CZC^T\epsilon. \tag{9.13}$$

No error accumulation can take place. Indeed, if $\Delta t \to 0$ one has $h(x_{n+1}) \to 0$, as desired.

For future reference we identify the term $h(x_n)/\Delta t$ as a *compensation term.* This name is motivated by the observation that $h(x_n)$ represents not only the value of the constraint evaluated in x_n but at the same time represents the constraint violation. The presence of $h(x_n)/\Delta t$ in the expression for λ_n^d can be interpreted as the 'compensation' of errors made at time t_n. Note however, that this compensation term is not added in a heuristic manner but instead follows from a mathematical derivation.

The expression for the discrete generalized Lagrange multiplier λ_n^d is useful in combination with other integration routines, and for nonlinear constraints as well. For this define on the intervals $[t_n, t_{n+1}]$, $n \in \mathbb{Z}_n$,

$$\lambda_n^d = -(C(x)Z(x)C^T(x))^{-1}\left(G(x)(f(x)+g(x)v) + \frac{h(x_n)}{\Delta t}\right). \tag{9.14}$$

Note that now only the compensation term is kept constant on an interval $[t_n, t_{n+1}]$. All other functions are to be evaluated in the points needed by the numerical method that is applied. We would like to emphasize that the use of the discrete generalized Lagrange multiplier λ_n^d does not yield numerically stiff equations when the time-step is reduced. This can be seen from (9.10). In that equation the Lagrange multiplier is multiplied again with Δt. As a result, the term Δt in the denominator of the compensation term is canceled.

We will discuss the Forward-Euler method, a class of Runge-Kutta-2 methods, and the Runge-Kutta-4 method. The formula for the Forward-Euler method has already been used in the analysis above. Assume we want to solve the differential equation $\frac{dx}{dt} = z(x)$, where z satisfies a Lipschitz condition [134]. Then a class of Runge-Kutta-2 methods is given by [2]:

$$x_{n+1} = x_n + \Delta t\left((1-\frac{1}{2\alpha})z(x_n) + \frac{1}{2\alpha}z(x_n + \alpha\Delta t\, z(x_n))\right), \tag{9.15}$$

with $\alpha \in (0,1)$. This class of Runge-Kutta-2 methods gives $\mathcal{O}((\Delta t)^2)$ accurate methods. The Runge-Kutta-4 method we will use is given by [2]:

$$x_{n+1} = \frac{1}{6}(V_1 + 2V_2 + 2V_3 + V_4), \tag{9.16}$$

where $V_1 = z(x_n)$, $V_2 = z(x_n + \frac{1}{2}\Delta t V_1)$, $V_3 = z(x_n + \frac{1}{2}\Delta t V_2)$, $V_4 = z(x_n + \Delta t V_3)$. Runge-Kutta-4 is an $\mathcal{O}((\Delta t)^4)$ accurate method. Let k denote the order of the ODE-solver and let $'0'$ denote machine zero.

Theorem 9.3.3 *Stable numerical integration of DAEs* ([38]). Consider the system in proposition 7.6.1 (*ii*). Let assumption 9.3.1 hold. Take the discrete Lagrange multiplier λ_n^d as in (9.14). Then the system can be solved in a stable manner with Forward-Euler ($k = 1$), the class of Runge-Kutta-2 methods ($k = 2$) in (9.15), and the Runge-Kutta-4 method ($k = 4$) in (9.16). Furthermore, if the discrete Lagrange multiplier λ_n^d is solved with sufficient accuracy there holds $h(x_n) =' 0' + \mathcal{O}((\Delta t)^{k+1})$. $\lhd$

In case that the initial conditions violate the constraints, evaluation of $h(x_{n+1})$ using the formulas for Forward-Euler and λ_n^d gives

$$h(x_{n+1}) = h(x_n - Z(x_n)C^T(x_n)(C(x_n)Z(x_n)C^T(x_n))^{-1}h(x_n) + \mathcal{O}(\Delta t)). \tag{9.17}$$

In this formula the first step of a Newton-Raphson iteration process can be recognized. For other ODE-solvers the situation is more complicated as intermediate points enter the discussion. Simulation results, however, clearly show that even for large initial violations the error is largely reduced within a few time-steps in case of a higher-order ODE-solver (see section 9.4).

It has been shown that the instability phenomenon already occurs for systems where $r_0 = 1$. For this case a remedy was given: use of the discrete Lagrange multiplier λ_n^d and standard ODE-solvers give numerical methods that offer error reduction and avoidance of error accumulation in one single algorithm. No need exists to stabilize the integration process any further. The results can be extended to systems with higher relative degree, but the results derived so far will be sufficient for the application we have in mind: constrained mechanical systems.

9.4 Bilaterally constrained mechanical systems

Before we can apply the results to a bilaterally constrained mechanical system we need some results and representations. Consider the system in (9.1), but now in a first-order formulation.

$$\begin{aligned} \tfrac{dx_1}{dt} &= x_2, \\ \tfrac{dx_2}{dt} &= -M(x_1)^{-1}N(x_1, x_2) + M(x_1)^{-1}v + M(x_1)^{-1}G^T(x_1)\lambda, \\ 0 &= \phi(x_1), \\ 0 &= G(x_1)x_2. \end{aligned} \tag{9.18}$$

In this formulation the fourth equation is redundant if the initial conditions are consistent with the constraints, but its presence is essential for our purposes. First we give the analogue of proposition 8.4.1 for the overdetermined system in (9.18). It can be seen that the result in proposition 9.4.1 below is similar to the result in proposition 8.4.1, apart from the presence of the Lagrange multiplier μ. From the proof of the proposition it follows that $\mu = 0$, but

the importance of the proposition given below lies in its use to obtain numerical solutions.

Proposition 9.4.1 *Bilaterally constrained mechanical systems.* Let $(x_{1,0}, x_{2,0})$ satisfy the constraints. Let the system in (9.18) satisfy the conditions of proposition 8.4.1. Then the following conditions are equivalent.

(*i*) $\exists u \in \mathbb{U}(\mathbb{R}_+, \mathbb{R}^m)$ such that $(\underline{x}_1, \underline{x}_2)$ is a trajectory of the system (9.18) in $\mathcal{M}_0$ with $(\underline{x}_1(0), \underline{x}_2(0)) = (x_{1,0}, x_{2,0})$,

(*ii*) $\exists v \in \mathbb{U}(\mathbb{R}_+, \mathbb{R}^m), \mu \in \mathbb{U}(\mathbb{R}_+, \mathbb{R}^p), \lambda \in \mathbb{U}(\mathbb{R}_+, \mathbb{R}^p)$ such that $(\underline{x}_1, \underline{x}_2)$ is a trajectory of the system

$$\begin{aligned}
\frac{dx_1}{dt} &= x_2 + G^T(x_1)\mu, \\
\frac{dx_2}{dt} &= -M(x_1)^{-1}N(x_1, x_2) + M(x_1)^{-1}v + M(x_1)^{-1}G^T(x_1)\lambda, \\
\phi(x_1) &= 0, \\
G(x_1)x_2 &= 0,
\end{aligned}$$

in $\mathcal{M}_0$ with $(\underline{x}_1(0), \underline{x}_2(0)) = (x_{1,0}, x_{2,0})$,

(*iii*) $\exists v \in \mathbb{U}(\mathbb{R}_+, \mathbb{R}^m)$ such that $(\underline{x}_1, \underline{x}_2)$ is a trajectory of the system

$$\begin{aligned}
\frac{dx_1}{dt} &= x_2 + G^T(x_1)\mu, \\
\frac{dx_2}{dt} &= -M(x_1)^{-1}N(x_1, x_2) + M(x_1)^{-1}v + M(x_1)^{-1}G^T(x_1)\lambda, \\
\mu &= -(G(x_1)G^T(x_1))^{-1}G(x_1)x_2, \\
\lambda &= -(G(x_1)M^{-1}(x_1)G^T(x_1))^{-1} \cdot \\
&\qquad \left(G(x_1)M^{-1}(x_1)(v - N(x_1, x_2) + \frac{dG}{dt}(x_1)x_2) \right),
\end{aligned}$$

in $\mathcal{M}_0$ with $(\underline{x}_1(0), \underline{x}_2(0)) = (x_{1,0}, x_{2,0})$. ◁

A more general version of the above proposition is given in [38], but the present formulation suffices for our purposes. For the case where $\frac{\partial G(x_1)}{\partial x_1}x_2 = 0$ the result in proposition 9.4.1 follows directly from proposition 7.6.1. For this first introduce fictitious controls u_1 to obtain $\frac{dx_1}{dt} = x_2 + u_1$. Apply proposition 7.6.1 with $Z(x) = diag(I, M(x_1))$. Next, set the controls in the differential equation for $\frac{dx_1}{dt}$ to zero again. The resulting set of differential equations contains a vector of Lagrange multipliers (μ, λ). It has been this observation that motivates the presence of the term $G^T(x_1)\mu$ in the proposition above. In [64] the formulation in proposition 9.4.1 (*ii*) can also be found. There the term $G^T(x_1)\mu$ is added to a given representation. (In case $\frac{\partial G(x_1)}{\partial x_1}x_2 \neq 0$ application of proposition 7.6.1 introduces a second Lagrange multiplier in the differential equation for x_1, next to μ. This leads to another overdetermined representation. This will not be discussed in this thesis.)

When the continuous-time equations in proposition 9.4.1 are discretized, the resulting formulations are not equivalent anymore. The idea, again, is to obtain expressions for the discrete generalized Lagrange multipliers by first discretizing the continuous set of equations. If we treat the ODE in an explicit manner and the constraint equations in an implicit manner then the expressions for the discrete Lagrange multipliers on an interval $[t_n, t_{n+1}]$

become

$$\begin{aligned}
\mu_n^d &= -(G(x_1)G^T(x_1))^{-1}(G(x_1)x_2 + \frac{\phi(x_1(t_n))}{\Delta t}), && (9.19)\\
\lambda_n^d &= -(G(x_1)M^{-1}(x_1)G^T(x_1))^{-1}\cdot && (9.20)\\
&\quad \left(G(x_1)M^{-1}(x_1)(v - N(x_1,x_2) + \frac{dG}{dt}(x_1)x_2) + \frac{G(x_1(t_n))x_2(t_n))}{\Delta t}\right).
\end{aligned}$$

We have now obtained two compensation terms, namely $\frac{\phi(x_1(t_n))}{\Delta t}$ and $\frac{G(x_1(t_n))x_2(t_n)}{\Delta t}$.

Next we state our main result with respect to stable numerical integration of mechanical systems subject to equality state-space constraints. The proof is a direct consequence of theorem 9.3.3 and is therefore omitted.

Theorem 9.4.2 *Stable numerical integration of bilaterally constrained mechanical systems.* Consider the system in proposition 9.4.1 (*ii*). Let assumption 9.3.1 hold. Take the discrete Lagrange multipliers μ_n^d and λ_n^d from formulations (9.19) and (9.20). Then the system can be solved in a stable manner with Forward-Euler ($k = 1$), the class of Runge-Kutta-2 methods ($k = 2$), and Runge-Kutta-4 ($k = 4$). Furthermore, if the discrete Lagrange multipliers μ_n^d and λ_n^d are solved with sufficient accuracy there holds: $\phi(x(t_n)) = {}'0' + \mathcal{O}((\Delta t)^{k+1})$, $G(x_1(t_n))x_2(t_n) =' 0' + \mathcal{O}((\Delta t)^{k+1})$. ◁

We will illustrate the theory with respect to stable numerical integration with the aid of the simple mechanical system of example 1.2.3. To avoid excessive use of indices, let the positions be denoted by y_i and the velocities by z_i, $i = 1, 2$. Take $r = 1$. From proposition 9.4.1 we obtain the following equations of motion.

$$\begin{aligned}
\frac{dy_1}{dt} &= z_1 + G_1^T(y)\mu,\\
\frac{dy_2}{dt} &= z_2 + G_2^T(y)\mu,\\
\frac{dz_1}{dt} &= G_1^T(y)\lambda, && (9.21)\\
\frac{dz_2}{dt} &= -\mathrm{g} + y_1 + G_2^T(y)\lambda,\\
0 &= y_1^2 + y_2^2 - 1,\\
0 &= 2y_1z_1 + 2y_2z_2,
\end{aligned}$$

with $G(y) = [G_1(y)\ G_2(y)] = [2y_1\ 2y_2]$.

In the simulation studies described here the following combinations of Lagrange multipliers are used (with simplified notation):

Combination A: $\lambda = -(GM^{-1}G^T)^{-1}(GM^{-1}(v - N + \frac{dG}{dt}x_2))$ and $\mu = 0$, which is the conventional approach to simulation, and

Combination B: the discrete Lagrange multipliers μ_n^d and λ_n^d, which is our approach to simulation.

The simulation results reported in this section all use a Runge-Kutta-Merson method from the NAG library [105]. Unless stated otherwise the integration time-step Δt was fixed and chosen as $\Delta t = 1/100$, and the integration interval as $[0, 1]$. Before performing the error analysis with respect to constraint violation, the time-step Δt used during a particular simulation was checked to be within the stability region of the numerical method. For this three runs where conducted with step sizes $2\Delta t$, Δt and $\frac{1}{2}\Delta t$, respectively. For the Runge-Kutta-4 method one expects a theoretical order 16, being 2^4. This was checked using the formula

$$Q(\Delta t) := \left| \frac{x_n(n \cdot \Delta t) - x_{\frac{n}{2}}(\frac{n}{2} \cdot 2\Delta t)}{x_{2n}(2n \cdot \frac{\Delta t}{2}) - x_n(n \cdot \Delta t)} \right|. \tag{9.22}$$

where x_n denotes the numerical approximation of a solution of the differential equation $\frac{dx}{dt} = f(x)$. In practise (9.22) can be used to obtain an indication whether the integration step size has been taken small enough. For all simulation results reported here, the value of $Q(\Delta t)$ was close to the expected value of 16.

First we discuss the case where all initial conditions are in agreement with the constraint equations. Second, we introduce initial errors with respect to the position constraint. Third, we treat the case where both the position and the velocity constraint are not initially satisfied.

We start with the case were all initial conditions are in agreement with the constraint equations. At time $t = 0$ we take $(y_1, y_2) = (1, 0)$ and $(z_1, z_2) = (0, 0)$. First we take combination A, corresponding to the conventional approach to simulation (but without additional stabilization) as discussed in section 9.2. The constraint violations are depicted in figure 9.2. Figure 9.3 depicts the simulation result when use is made of the discrete Lagrange multipliers μ_n^d and λ_n^d (combination B). If we now compare the error level obtained from a simulation by the conventional way and the error level obtained by using the theory developed here it follows that the latter formulation reduced the error in the velocity constraint violation by a factor 10. An even larger reduction is obtained for the position constraint violation. A factor 100 is gained with respect to the violation of the position constraint. The oscillatory behaviour of the constraint violations is due to the oscillatory behaviour of the ball.

Second, we discuss the case where the initial conditions are not correct. We start with $(y_1, y_2) = (2, 0)$ and $(z_1, z_2) = (0, 0)$. So, the position constraint is not initially satisfied. This amount of constraint violation may not be very likely in real world applications where initial conditions may be obtained from sensor information. Here it serves to show the

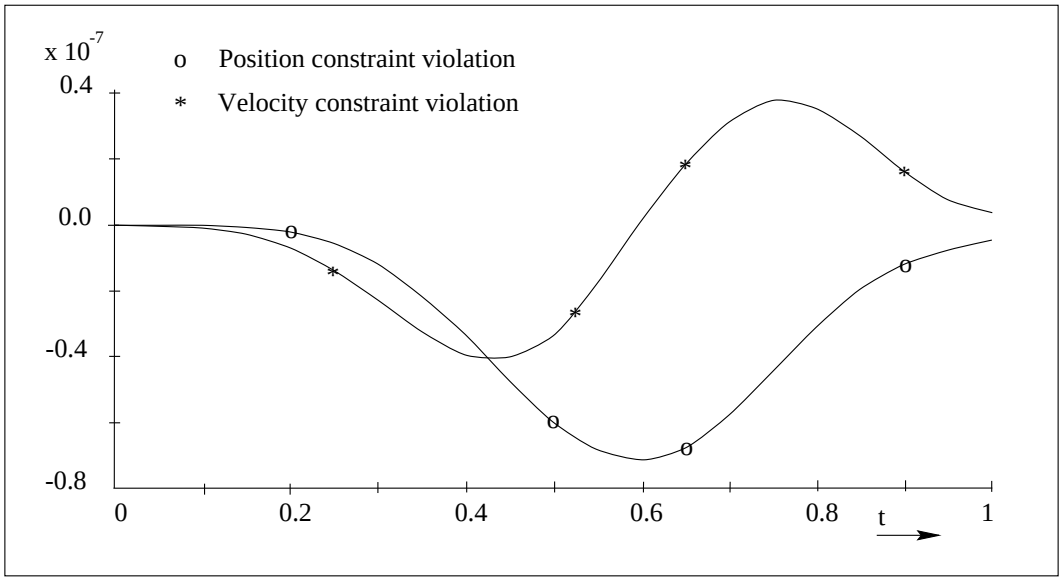

Figure 9.2: Simulation with combination A: consistent initial conditions.

difference between simulations performed in the conventional manner and simulations performed with the full model as developed in this chapter. The ball remains on a circle with radius 2 if we use combination A [38, figure 5]. The error due to the initial conditions is not reduced. If we perform simulations using combination B the error behaviour depicted in figure 9.4 is obtained. Here the violations at the first three times-steps are not shown as they do not fit the scale. Even though the position constraint is nonlinear, the initial error is greatly reduced within a small number of integration steps and the error behaviour is very close indeed to the one with correct initial conditions depicted in figure 9.3.

Finally, we discuss the case where the initial conditions on position level and on velocity level are violated. Take $(y_1, y_2) = (1, 0.01)$ and $(z_1, z_2) = (1, 0.1)$. The position constraint violation equals $\phi(y) = 10^{-4}$, which is quite small. The velocity constraint violation equals $G(y)z = 2.002$, which is significant. This test case was used to demonstrate the instability in example 9.2.1. Figure 9.1 depicts the numerical behaviour of the ball using combination A. The ball is not on the unit circle, and the position constraint violation increases with time. When we use combination B the ball does stay on the unit circle (apart from the first time-steps). The error behaviour is not shown, but is similar to the one depicted in figure 9.4.

The case where the initial conditions are consistent was also used to verify the order of the constraint violation, as predicted by theorem 9.3.3. Runge-Kutta-Merson uses a fourth order Runge-Kutta integration method. Hence by theorem 9.3.3 we expect to see a factor 32, being 2^5, when we reduce the time-step Δt by a factor two and take the quotient of the constraint violations at the same discrete time instant. The results are depicted in tables 9.1 and 9.2, taken from [38]. The factors in the column $(\cdot/\cdot)$ are in agreement with the

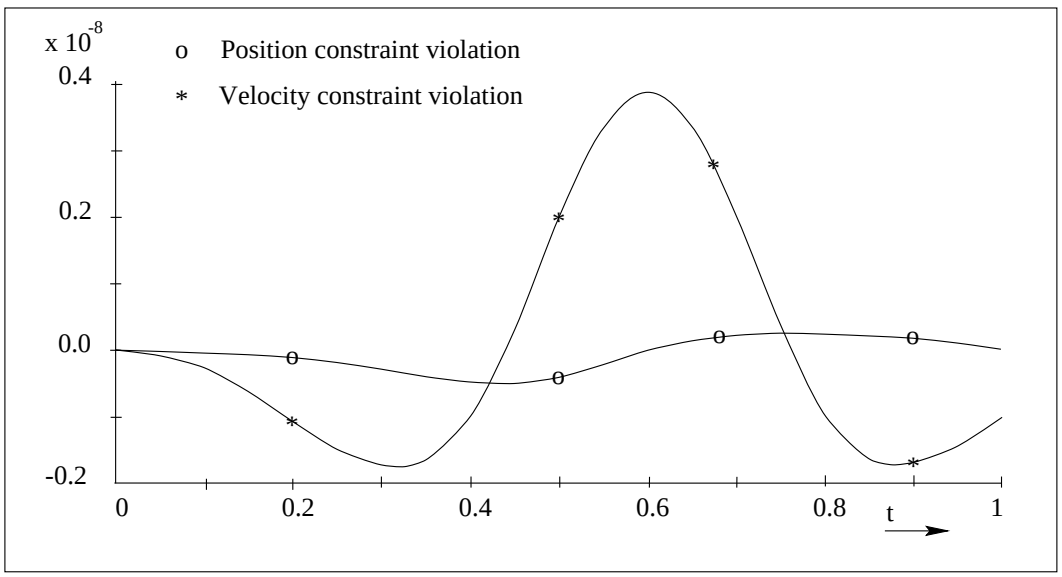

Figure 9.3: Simulation with combination B: consistent initial conditions.

theory developed here.

Table 9.1: Verification of the order of the position constraint error behaviour.

Time	$\Delta t = 0.0625$ Error	$\cdot/\cdot$	$\Delta t = 0.03125$ Error	$\cdot/\cdot$	$\Delta t = 0.015625$ Error
0.000	$0.0000E+0$	–	$0.0000E+0$	–	$0.0000E+0$
0.250	$-0.1107E-5$	29	$-0.3864E-7$	31	$-0.1258E-8$
0.500	$-0.4243E-5$	37	$-0.1143E-6$	35	$-0.3231E-8$
0.750	$-0.2459E-5$	25	$-0.9800E-7$	29	$0.3329E-8$
1.000	$0.5952E-6$	32	$0.1872E-7$	33	$0.5716E-9$

In this chapter we have so far discussed a simple mechanical system. Since the numerical method has been developed for the general model in (9.1), it is applicable to a large class of mechanical systems. The theory developed here has been used successfully to simulate a pair of two-link rigid manipulators in generalized joint coordinates. Several nonlinear constraints were superimposed upon the system. Simulations have been extended to longer periods of time, and no error accumulation took place. Simulation of controlled robot operations indicate that the values of the compensation terms are much smaller than the values of the (feedback) control. This implies that the numerical compensation terms are just that: they do not interfere with the control actions (see also [85] and the references therein). In [33, 36] the theory has been applied to a six degree of freedom robotic manipulator with gearbox flexibility. The study reported in [33] was also used to see whether the use of the Lagrange multipliers μ_n^d and λ_n^d would mean a significant increase in computation time. The simulation made use of a complex robotic manipulator model, including gearbox

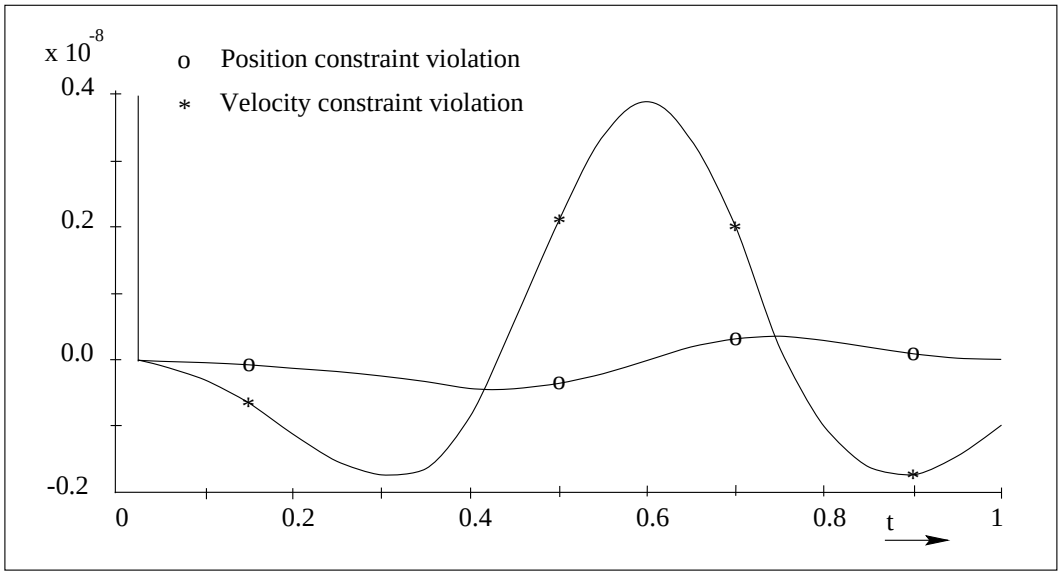

Figure 9.4: Simulation with combination B: incorrect initial conditions.

Table 9.2: Verification of the order of the velocity constraint error behaviour.

Time	$\Delta t = 0.0625$ Error	$\cdot/\cdot$	$\Delta t = 0.03125$ Error	$\cdot/\cdot$	$\Delta t = 0.015625$ Error
0.000	$0.0000E+0$	–	$0.0000E+0$	–	$0.0000E+0$
0.250	$-0.1357E-4$	31	$-0.4426E-6$	32	$-0.1397E-7$
0.500	$0.4853E-5$	12	$0.4155E-6$	25	$0.1695E-7$
0.750	$0.1598E-4$	60	$0.2682E-6$	58	$0.4659E-8$
1.000	$-0.9525E-5$	33	$-0.2919E-6$	33	$-0.8928E-8$

flexibility. The simulation lasted for a period of 100 simulated seconds. In the conventional approach to simulation the computation time was 30.71 seconds. With combination B the computation time was 31.58 seconds, which amounts to an increase of approximately 0.028% per simulated second. In [53, 148] the theory is used to obtain simulation results of a fully flexible robotic manipulator that is to operate in space. We refer to the above cited references for more simulation results. In each of these cases the simulation results were in agreement with the theory: no error accumulation took place and the correct dynamic behaviour was obtained. For simulations where the manipulator Jacobian matrix $J(q)$ is singular we refer to [54].

The numerical method we have obtained can be interpreted in a number of ways. The method may be viewed as a special case of a predictor/ corrector algorithm, where the predictor step and the corrector step are combined in one step. A similar remark holds for an interpretation as a projection method. From a control perspective, the compensation terms can be interpreted as proportional feedback for a first-order system.

The theory can be extended to cover constraints $\phi(x,t) = 0$, i.e. constraints that depend explicitly on the time. These constraints arise for instance in dynamic path planning of robotic manipulators, and also in flight-path management of aeroplanes. Most of the theory remains valid [38], but in case of Runge-Kutta-4 there are additional requirements on higher derivatives of the constraints that need to be satisfied. If these requirements are not met then $\phi(x_{n+1}) = \mathcal{O}((\Delta t)^3)$ can be attained instead of the accuracy stated in theorems 9.3.3 and 9.4.2. This still leads to stable numerical simulation of constrained dynamical systems.

9.5 Unilaterally constrained mechanical systems

Up to now we have discussed numerical aspects of what is known as the constrained motion phase for robotic manipulators: the motion remains on the boundary set. In this section we will briefly discuss some numerical aspects of (un)controlled collisions with a single boundary set, based on practical experience. Other approaches that deal with the discontinuities in the velocity can be found in for instance [95] for general optimal control problems, and in [50, 94] for autonomous systems.

It is clear that there is a need for simulation tools that provide routines that can be used to simulate all the transitions and collisions discussed in this thesis. However, even for bilaterally constrained mechanical systems, commercial tools are not abundant and usually the numerical code that implements the transition rules and discontinuities due to uncontrolled collisions must be supplied by the user.

Our goal is again the support of real-time simulation and we actively pursue numerical methods that do not use iteration processes. In order to simulate unilaterally constrained dynamical systems, one needs to know whether or not there will be a jump in the state, and if so, how this jump is made. The simulation of contact uses the model in proposition 9.4.1 together with the discrete Lagrange multipliers μ_n^d and λ_n^d, the collision maps, and the transition rules. Note that the transition rules from $\mathcal{N}^*$ to $\mathcal{M}_g$ should be based on the value of λ only. The Lagrange multiplier μ is present only to facilitate numerical simulation and has no physical meaning.

All contact and release sets can be computed off-line using the results from chapters 6 and 7. Using these sets, explicit expressions for the collision maps can be made off-line as well. During simulation, once the contact point is known, a simply check followed by one function evaluation then suffices to obtain the desired results. Due to discretization however, there remain a number of problems with respect to simulating uncontrolled and controlled contact. The problem with (un)controlled contact is the approximation of the time that contact with the boundary set is made. The problem with controlled contact is the validation of the transition rules for contact and release. Since real-time simulation often involves trade-off between accuracy and computation time, we will introduce some parameters that should be available to the user of a simulation program to customize

simulation to his/her specific needs.

First we discuss the problem of determining the time that a trajectory makes contact with the boundary set. Assume that at time t_n there holds $\phi(x(t_n)) > 0$. If contact is made in the interval (t_n, t_{n+1}) then this can be detected only if

$$\phi(x(t_n)) \cdot \phi(x(t_{n+1})) \leq 0. \tag{9.23}$$

This also implies that if a controlled contact and a controlled release take place in the interval (t_n, t_{n+1}), (and the motion stays in $\mathcal{N}^*$ for a small period of time), this can be detected numerically only if the time-step is small enough. We assume that this is the case.

Depending on the characteristics of the numerical solution, it may well be that $\phi(x(t_{n+1}))$ is significantly smaller than zero. This implies that even with a fixed step-size, so called step back is necessary. Step back means that the time-step Δt is adjusted and the simulation is started again from time t_n. This means that the same dynamic equations are solved again. And if the new estimate of $x(t_{n+1})$ is not satisfactory, again step back is necessary. Clearly such an iterative procedure may increase computation time significantly.

Our approach is based on the method of false position or *regula falsi* [134]. Assume that $\phi(x(t_n)) > 0$ and $\phi(x(t_{n+1})) \leq 0$. We want to establish the time-instant t^* where the trajectory makes contact with the boundary set. Since we assume stable numerical integration, the numerical approximated trajectory and the analytical trajectory are related through the accuracy of the numerical ODE solver that is used. This means that it makes sense to search for a time instant $\hat{t}$ that is close, but not necessarily equal to t^*. Our approach is to use a linear interpolation on the interval $[t_n, t_{n+1}]$. This gives

$$\phi(x(t_n + t_i)) = \frac{\phi(x(t_{n+1})) - \phi(x(t_n))}{\Delta t} \cdot t_i + \phi(x(t_n)) \tag{9.24}$$

with $t_i \in [0, \Delta t]$. Now we choose to adapt the step-size as

$$\widehat{\Delta t} := \left| \frac{\phi(x(t_n))}{\phi(x(t_{n+1})) - \phi(x(t_n))} \right| \cdot \Delta t. \tag{9.25}$$

This gives that approximately $\phi(x(t_n + \widehat{\Delta t})) = 0$ from (9.24). This approximation of the contact point may already be acceptable to the user of the simulation program. And using this value of $\widehat{\Delta t}$, a linear interpolation with the numerical value of the velocities at times t_n and t_{n+1} may give an acceptable value of the velocity component at contact. But we can go on by restarting the simulation from time t_n by using $\widehat{\Delta t}$ as the new time-step. Now the obtained numerical approximation is checked again to see if $\phi(x(t_n + \widehat{\Delta t})) \leq 0$. If this holds then the procedure outlined above can be repeated until a point is reached for which the position constraint is not satisfied. The approximation of the time of contact, $\hat{t}$, is now set equal to $t_n + \widehat{\Delta t}$, where $\widehat{\Delta t}$ is the last update such that $\phi(x(t_n + \widehat{\Delta t})) \leq 0$. Next, the time $t_n + \widehat{\Delta t}$ is also used to approximate the velocity vector at the time of contact.

Subsequently we set the time-step to its old value Δt again as the time-step $\widehat{\Delta t}$ may be too small to maintain real-time simulation. A parameter γ should be introduced to set an a priori bound on the number of iterations. The trade off between accuracy and computation time is then put in the hands of the user.

Even though the procedure outlined above is simple, and is motivated by the real-time application we have in mind, it still uses an iteration process. If the time needed to execute the procedure violates real-time simulation, the initial linear interpolation step in (9.24) may be replaced by a higher-order interpolation method using information at velocity level. Another promising approach is to use in the iteration process itself the equation $M(y)\frac{d^2y}{dt^2} = u$ instead of the one in (9.1), i.e. set $N = 0$. This choice is motivated by the fact that, by proposition 8.3.3, the contact set is independent of the functions M and N in (9.1), and the importance of the function M in calculating the impulse (see equations 8.19 and 8.20). This is still an active research area for real-time simulations of unilaterally constrained robotic manipulators.

Next it must be determined whether we are dealing with a collision or not. In case plastic collisions are considered, after detection of contact, the Lagrange multipliers are activated and the motion remains on the surface. When elastic collisions are possible, there is another numerical problem since $\frac{d\phi}{dt}(x(t^*))$ will in general not be zero numerically. This problem is similar to the problem in bilaterally constrained mechanical systems when the reaction force is approximately zero. A heuristic approach is the following. If at a contact point one has that $\nu \leq \frac{d\phi}{dt}(x(t^*)) \leq 0$, for a user defined value of the parameter ν, then the motion is assumed to proceed in $\mathcal{N}^*$. If $\frac{d\phi}{dt}(x(t^*)) < \nu (\leq 0)$ then the motion is to proceed by the use of an uncontrolled collision map. To detect release from the surface a check is made whether or not $\lambda_n^d < 0$. If this inequality holds true, then the Lagrange multiplier is deactivated. Since the compensation terms are usually much smaller then the value of λ_n^d itself, these compensation terms have no influence on this decision [85].

There remains a problem when the desired (simulated) contact force during motion in $\mathcal{N}^*$ is very small. But then controller design is also difficult: the slightest deviation from the desired path will mean that release takes place, although control is aimed at maintaining contact. This is the reason that in practice a certain amount of (constant) normal force on the surface is chosen.

A simulation of elastic collisions of a controlled robotic manipulator with a rigid wall is given in figure 8.2. During the simulation run the elasticity coefficient was $\epsilon = 1$. The parameter ν was set to zero since all contact is taken as an uncontrolled collision. Usually one adjustment of the time-step was sufficient. As already remarked in section 8.5, figure 8.2 reveals that even in the case of fully elastic collisions, the limit behaviour of the fully elastic collisions can be similar to the case where the elasticity parameter satisfies $0 < \epsilon < 1$. This is due to the applied control in the time interval between subsequent collisions. Further research on this subject is necessary, but for all practical purposes, if the velocity jump or the length of the time-interval between collisions is on the level of machine zero, plastic

collisions and elastic collisions can not be distinguished from each other numerically. The decision is then made to proceed the motion on the constraint surface.

9.6 Conclusions

We have proposed a numerical technique that can be used during real-time simulation of bilaterally constrained mechanical systems. A drift-off problem arises when simulations are performed in a conventional manner. Our solution to avoid drift-off involved solving the combination of differential and algebraic equations only after they have been discretized. It was shown that the resulting set of discrete equations differs from the set of equations obtained if the conventional approach to constrained dynamical systems is applied. Use of the discrete equations here derived, in combination with standard (explicit) ODE-solvers gives numerical methods that allow stable numerical integration. Features of the method are that no additional iteration process is necessary, and decreasing the time-step does not yield numerically stiff equations if the unconstrained system is not stiff. The derived numerical method is robust with respect to errors in the initial conditions and is stable with respect to errors made during the integration process. Simulation studies were performed based on the theory developed here. All simulation results reported here and in the literature were in agreement with the theoretical results derived in this chapter.

Simulation of unilaterally constrained dynamical systems is still a very active research area, also at NLR. We have outlined a procedure aimed at real-time simulation. The results have been applied successfully to simulate complex robotic manipulators subject to inequality constraints.

Appendix 9.A: Proofs

Proof of lemma 9.3.2:
Use of the equality in (9.10) gives

$$\begin{aligned} h(x_{n+1}) &= h(x_n + \Delta t(f(x_n) + g(x_n)v_n + g(x_n)ZC^T\lambda_n^d) \\ &= h(x_n) + \Delta tG(f(x_n) + g(x_n)v_n + g(x_n)ZC^T\lambda_n^d) \\ &= h(x_n) + \Delta tG(f(x_n) + g(x_n)v_n) + \Delta tg(x_n)CZC^T\lambda_n^d. \end{aligned}$$

Now require that the equality in (9.11) holds, and solve the resulting equation for λ_n^d. This gives the desired result. $\triangleleft$

Proof of theorem 9.3.3:
We will prove the statement for Forward-Euler and Runge-Kutta-2. The proof in case of Runge-Kutta-4 runs similar to the proof in case of Runge-Kutta-2.

Proof of the statement for Forward-Euler: The proof runs similar to the analysis in (9.7).

$$\begin{aligned}
h(x_{n+1}) &\overset{FE}{=} h(x_n + \Delta t(f(x_n) + g(x_n)v_n + g(x_n)Z(x_n)C^T(x_n)\lambda_n^d) \\
&\overset{Taylor}{=} h(x_n) + G(x_n)\Delta t(f_n + g(x_n)v_n + g(x_n)Z(x_n)C^T(x_n)\lambda_n^d) + \mathcal{O}((\Delta t)^2) \\
&= h(x_n) + \Delta t G(x_n)(f_n + g(x_n)v_n) - \Delta t G(x_n)g(x_n)Z(x_n)C^T(x_n) \cdot \\
&\quad ((C(x_n)Z(x_n)C^T(x_n))^{-1}G(x_n)(f(x_n) + g(x_n)v_n) - \frac{h(x_n)}{\Delta t}) \\
&\quad + \mathcal{O}((\Delta t)^2) \\
&= h(x_n) - h(x_n) + \mathcal{O}((\Delta t)^2) \\
&= '0' + \mathcal{O}((\Delta t)^2).
\end{aligned}$$

Proof of the statement for Runge-Kutta-2
Differentiation of the constraint $h(x) = 0$ gives $G(x)\frac{dx}{dt} = 0$. First consider solving $\frac{dx}{dt} = z(x)$. Denote $G_x := \frac{\partial G}{\partial x}$, and $z_x := \frac{\partial z}{\partial x}$. Differentiating the constraint again gives (omitting the arguments) $z^T G_x z + G z_x z = 0$. Using the proof in [2] that Runge-Kutta-2 is $\mathcal{O}((\Delta t)^2)$ accurate, gives (independent of α)

$$\begin{aligned}
P(x_{n+1}) &= P(x_n + \Delta t((1 - \frac{1}{2\alpha}z(x_n) + \frac{1}{2\alpha}z(x_n + \alpha\Delta t z(x_n)) + \mathcal{O}((\Delta t)^3)) \\
&= P(x_n + \frac{1}{2}\Delta t z(x_n) + \frac{1}{2}(\Delta t)^2 z_x(x_n)z(x_n) + \mathcal{O}((\Delta t)^3))) \\
&= P(x_n) + G(x_n)(\frac{1}{2}\Delta t z(x_n) + \frac{1}{2}(\Delta t)^2 z_x(x_n)z(x_n) + \mathcal{O}((\Delta t)^3))) \\
&\quad \frac{1}{2}(\frac{1}{2}\Delta t z(x_n) + \frac{1}{2}(\Delta t)^2 z_x(x_n)z(x_n))^T G_x(x_n)(\frac{1}{2}\Delta t z(x_n) + \\
&\quad \frac{1}{2}(\Delta t)^2 z_x(x_n)z(x_n)) + \mathcal{O}((\Delta t)^3)) \\
&= P(x_n) + \Delta t G(x_n)z(x_n) + \frac{1}{2}(\Delta t)^2(z^T(x_n)G_x(x_n)z(x_n) + \\
&\quad G(x_n)z_x(x_n)z(x_n) + \mathcal{O}((\Delta t)^3)).
\end{aligned}$$

Using again that Runge-Kutta-2 is a $\mathcal{O}((\Delta t)^2))$ accurate method, substitution of $z = f + gv + gZC^T\lambda_n^d$ and use of the expressions for λ_n^d now gives that $P(x_{n+1}) = \mathcal{O}((\Delta t)^3)$. $\lhd$

Proof of proposition 9.4.1:
First observe that if $\mu = 0$ all statements follow from proposition 8.4.1, where the system is now in a first-order form. We are done if we can prove that in formulations (*ii*) and (*iii*) there holds $\mu = 0$. First consider formulation (*ii*). Premultiplication of the first equation with $G(x_1)$ gives $G(x_1)\frac{dx_1}{dt} = G(x_1)x_2 + G(x_1)G^T(x_1)\mu$. Using the constraint equations gives $G(x_1)G^T(x_1)\mu = 0$. From the assumptions it now follows that $\mu = 0$. Next consider formulation (*iii*). Substitution of the expression of μ into the first equation gives $G(x_1)\frac{dx_1}{dt} = G(x_1)x_2 - G(x_1)x_2 = 0$. Since the initial conditions satisfy the position constraint it follows that $\phi(x_1) = 0$. It is now immediate that $\mu = 0$. $\lhd$